A Cognitive Serverless Framework for the Cloud-Edge Continuum

D4.2 COGNIT Serverless Platform - Scientific Report - b

Version 1.0

30 April 2024

Abstract

COGNIT is an AI-enabled Adaptive Serverless Framework for the Cognitive Cloud-Edge Continuum that enables the seamless, transparent, and trustworthy integration of data processing resources from providers and on-premises data centers in the cloud-edge continuum, and their automatic and intelligent adaptation to optimise where and how data is processed according to application requirements, changes in application demands and behaviour, and the operation of the infrastructure in terms of the main environmental sustainability metrics. This document describes the research and development carried out in WP4 “AI-enabled Distributed Serverless Platform and Workload Orchestration” during the Second Research & Innovation Cycle (M10-M15), providing details on the status of a number of key components of the COGNIT Framework (i.e. Cloud-Edge Manager and AI-Enabled Orchestrator) as well as reporting the work related to supporting Energy Efficiency Optimization in the Multi-Provider Cloud-Edge Continuum.

This project is funded by the European Union’s Horizon Europe research and innovation programme under Grant Agreement 101092711 – SovereignEdge.Cognit

Deliverable Metadata

Project Title:	A Cognitive Serverless Framework for the Cloud-Edge Continuum
Project Acronym:	SovereignEdge.Cognit
Call:	HORIZON-CL4-2022-DATA-01-02
Grant Agreement:	101092711
WP number and Title:	WP4. AI-enabled Distributed Serverless Platform and Workload Orchestration
Nature:	R: Report
Dissemination Level:	PU: Public
Version:	1.0
Contractual Date of Delivery:	31/03/2024
Actual Date of Delivery:	30/04/2024
Lead Author:	Monowar Bhuyan (UMU) & Paul Townend (UMU)
Authors:	Malik Bouhou (CETIC), Simon Bonér (UMU), Aritz Brosa (Ikerlan), Zhou Zhou (UMU), Idoia de la Iglesia (Ikerlan), Sébastien Dupont (CETIC), Aitor Garciandia (Ikerlan), Joan Iglesias (ACISA), Tomasz Korniluk (Phoenix), Johan Kristiansson (RISE), Antonio Lalaguna (ACISA), Marco Mancini (OpenNebula), Alberto P. Martí (OpenNebula), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Daniel Olsson (RISE), Per-Olov Östberg (UMU), Goiuri Peralta (Ikerlan), Samuel Pérez (Ikerlan), Bruno Rodríguez (OpenNebula), Juan José Ruiz (ACISA), Kaja Swat (Phoenix), Thomas Ohlson Timoudas (RISE), Iván Valdés (Ikerlan), Constantino Vázquez (OpenNebula), David Carracedo (OpenNebula), Ignacio M. Llorente (OpenNebula), Victor Palma (OpenNebula), Michal Opala (OpenNebula), Pavel Czerny (OpenNebula), Jackub Walczak (OpenNebula).
Status:	Submitted

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	23/04/2024	Draft	Initial Draft
0.2	25/04/2024	Peer-Reviewed	Reviewed Draft
1.0	30/04/2024	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.1	24/04/2024	Antonio Álvarez (OpenNebula)
0.1	25/04/2024	Goiuri Peralta (Ikerlan)

Summary of Changes from Previous Versions

First Version of Deliverable D4.2

[1] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted, and Approved.

Executive Summary

This is the second “COGNIT Serverless Platform - Scientific Report” that has been produced in WP4 “AI-enabled Distributed Serverless Platform and Workload Orchestration”. It describes in detail the progress of the software requirements that have been active during the Second Research & Innovation Cycle (M10-M15) in connection with these main components of the COGNIT Framework:

Cloud-Edge Manager

- **SR4.3** Serverless Runtime Deployment:

  Deploy Serverless Runtime as Virtualized Workloads (e.g. Containers or VMs/microVMs) on the cloud-edge infrastructure.

- **SR4.4** Metrics, Monitoring, Auditing:

  Edge-Clusters monitoring, Serverless Runtimes metrics collection and continuous security assessment.

AI-Enabled Orchestrator

- **SR5.1** Building Learning Model:

  Implement AI/ML model based on collected metrics from Edge Cluster entities and serverless runtimes deployed across the distributed cloud-edge continuum.

- **SR5.2** Smart Deployment of Serverless Runtimes:

  Implement a Smart Workload Orchestrator (SWO) that exposes a REST API used by the Cloud-Edge Manager for requesting the deployment plans used for provisioning the Serverless Runtimes.

- **SR5.3** Scheduling Mechanisms:

  Implement a scheduler that will place the Serverless Runtimes on the Edge-Clusters resources according to the deployment plan provided by the AI-Enabled Orchestrator.

This deliverable has been released at the end of the Second Research & Innovation Cycle (M15), and will be updated with incremental releases at the end of each research and innovation cycle in M21, M27, and M33.

Table of Contents

Abbreviations and Acronyms

AE	Auto Encoder
AI	Artificial Intelligence
AI-O	AI-Enabled Orchestrator
API	Application Programming Interface
CPCA	Common Principal Component Analysis
DB	Database
DL	Deep Learning
FaaS	Function as a Service
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
IDEC	Improved Deep Embedded Clustering
IP	Internet Protocol
JSON	Javascript Object Notation
LSTM	Long Short-Term Memory
ML	Machine Learning
MSE	Mean Squared Error
MTS	Multivariate Time Series
OS	Operating System
QoS	Quality of Service
RAPL	Running Average Power Limit
REST	Representational State Transfer
SLA	Service Level Agreement
SLO	Service Level Objective
SVD	Single Value Decomposition
VM	Virtual Machine

1. Cloud-Edge Manager

The Cloud-Edge Manager is responsible for managing the cloud-edge continuum infrastructure and performing actions to manage the lifecycle of the different Serverless Runtimes, collecting their metrics and monitoring the infrastructure resources they use.

The main responsibilities of the Cloud-Edge Manager are, thus:

- Exposing through an API the operations for managing the cloud-edge continuum infrastructure (i.e., physical computational hosts, networks and storages across multi-cloud providers and edge locations) and managing the Serverless Runtimes, used by Device Client to offload functions.
- Monitoring both the cloud-edge infrastructure and the Serverless Runtimes to provide the AI-Enabled Orchestrator with information to implement automatic and intelligent adaptation for the placement of the Serverless Runtimes.
- Providing authentication and authorization mechanisms for accessing and securing resources such as physical hosts, virtual resources, networks, services, etc.

In this second development cycle the main work performed in the Cloud-Edge Manager relates to the deployment of the Serverless Runtime, with provided guides and improved mechanisms to report readiness of the service and addition of different metrics to aid in the intelligent orchestration of resources.

[SR4.3] Serverless Runtime Deployment

Description

As introduced in D4.1, Serverless Runtimes are modelled in the Cloud-Edge Manager as an OpenNebula OneFlow service, which in turn is composed of OpenNebula VM Templates and Images.

A Serverless Update guide[2] was created and distributed to support the need of the different use cases defining their own Serverless Runtime flavours, with specific software libraries related to their specific applications. The guide explains the actions necessary to either update an existing Serverless Runtime, or to clone an existing one and update the new copy.

The Serverless Runtime deployment relies on the OpenNebula contextualization mechanism. OpenNebula contextualization is the process by which a virtual machine (VM) can be dynamically configured and customised during its instantiation or runtime within an OpenNebula cloud environment. Contextualization allows administrators to automate the configuration of VMs, ensuring that they meet specific requirements and are properly integrated into the cloud environment. This mechanism is used in the Serverless Runtime deployment procedure to implement a number of functionalities:

[2] https://github.com/SovereignEdgeEU-COGNIT/serverless-runtime/wiki/How-to-update-sr-template-image

- Start of the Serverless Runtime service within the Virtual Machine. This is achieved using existing functionality[3] that triggers scripts at boot time. A script was developed to perform a series of automated steps:
    - clone the COGNIT Serverless Runtime repository.[4]
    - install dependencies of the Serverless Runtime service.
    - optionally, install flavour specific dependencies.
    - launch the Serverless Runtime service.

- The existing OpenNebula OneGate component[5] is used to push information about the VM having finished its boot procedure, setting a custom attribute (READY=yes) to the VM metadata, allowing the Provisioning Engine to report the Serverless Runtime as ready when this attribute is found. This is performed if a Virtual Machine metadata contains the REPORT_READY attribute. When set to YES, the OpenNebula contextualization packages (that are executed in the guest OS as part of its boot process) will report the VM to be READY to the OneGate client. This is useful for the OpenNebula OneFlow service to determine when a VM is in running state as the VM might be running from the perspective of the hypervisor, but still booting and or configuring important services, like the *one-context* daemon.

This existing feature showed a critical limitation in the COGNIT execution model. Having the READY=yes attribute in a Virtual Machine metadata means that the VM has been booted, however, it does NOT imply that the Serverless Runtime service is running. Hence, a high failure rate was detected from the Device Client at the time of offloading the first function, as usually the Serverless Runtime service takes time to be launched that needs to be accounted for. Therefore, an extension of the existing contextualization mechanism in OpenNebula has been implemented in this second cycle, to answer the need of the Provisioning Engine to report the Serverless Runtime as running when the Serverless Runtime Application initialization is finished.

Data model

Two new attributes (see Table 1.1) were added in the OpenNebula contextualization mechanism, contributed upstream. The contextualization mechanism checks if these attributes are defined in the VM metadata and acts according to its values.

Attribute Name	Description
READY_SCRIPT	When the variable is defined, the REPORT_READY functionality will only be used after the contents of the variable are successfully executed. It is useful to customise your appliance readiness. For example READY_SCRIPT="nc -vz localhost 8000" will only return 0 if the port 8000 is up.

[3] https://docs.opennebula.io/6.8/management_and_operations/references/template.html#context-section
[4] https://github.com/SovereignEdgeEU-COGNIT/serverless-runtime
[5] https://docs.opennebula.io/6.8/management_and_operations/multivm_service_management/onegate_usage.html

READY_SCRIPT_PATH	Similar to READY_SCRIPT but the script exists in the Guest filesystem and its path is what needs to be defined on the CONTEXT section. For example READY_SCRIPT_PATH=/usr/bin/echo.

Table 1.1: New contextualization attributes

This functionality is now being used on the Serverless Runtimes VM Templates to check if the Serverless Runtime Application running inside the VM has bound its port.

[SR4.4] Metrics, Monitoring, Auditing

Extensive work was carried out in this second development cycle to enrich the monitoring attributes gathered by OpenNebula and Prometheus in order to support the different scheduling decisions made by the AI-Enabled Orchestrator component. We will present the work grouped by the type of metric to be measured.

COGNIT Specific Metrics

As presented in D3.2 under the "[SR2.1] Secure and Trusted FaaS Runtimes section", the Serverless Runtime service exhibits a Prometheus Exporter, running by default on port 9100.

A new attribute is now available in the VM metadata, as seen in Table 1.2.

Attribute Name	Description
PROMETHEUS_EXPORTER	Prometheus exporter port. This is used by the OpenNebula Prometheus integration to scrap the Prometheus exporter running on the VM guest OS.

Table 1.2: New VM metadata attribute for COGNIT integration with Prometheus

If this attribute is defined, then the OpenNebula frontend will automatically add those VMs as scrapping targets of the Prometheus server running in the frontend. This is achieved by extending the current Prometheus integration, which relies on a datasource patch script which takes care of updating the Prometheus configuration file that holds the targets to scrap. In other words, it updates the Prometheus scraping endpoints (ie, what Prometheus can inspect to extract metrics) and that needs to be run manually. In a production OpenNebula deployment this is not a significant problem since the integration only covers hypervisor exporter scrapping rather than per VM exporter scrapping. Adding a hypervisor node to an OpenNebula cloud is a much more static and rare operation than creating a new VM, so in the context of the COGNIT project a more dynamic approach is needed, as it is not realistic to rely on manual update of the Prometheus datasources.

As part of the development of this functionality of dynamic scrapping endpoints for the OpenNebula Prometheus integration, two OpenNebula hooks[6] were created (hooks are scripts that can be associated with any resource state change or API call in OpenNebula), a VM state hook and a HOST state hook. Both hooks are intended to automatically run the data sources patch script every time a VM reaches the RUNNING state and a HOST reaches the MONITORED state.

[6] https://docs.opennebula.io/6.8/integration_and_development/system_interfaces/hook_driver.html#overview

The default patch data sources are also modified to include the addition of this new type of exporter. An excerpt of the config file can be found on *Figure 1.1*, showcasing IPv6 and IPv4 targets.

```python
- job_name: sr_exporter
  static_configs:
  - targets:
    - "[2001:67c:22b8:1::11]:9100"
    labels:
      vm_id: '1329'
  - targets:
    - "[2001:67c:22b8:1::7]:8787"
    labels:
      vm_id: '1323'
  - targets:
    - "[2001:67c:22b8:1::9]:8787"
    labels:
      vm_id: '1320'
  - targets:
    - "[2001:67c:22b8:1::8]:8787"
    labels:
      vm_id: '1317'
```

Figure 1.1 Prometheus Scrapper Configuration File

This new mechanism has been deployed on the COGNIT testbed and activated to the VM Template backing each of the use cases Serverless Runtime flavours.

Energy Metrics for Virtual Machines

As a continuation of the work done in the first development cycle and presented in D4.1, the Scaphandre integration in OpenNebula has been extended to allow for VM energy metric extraction, as well as the already performed hypervisor energetic metrics extraction.

Scaphandre[7] is a metrology agent dedicated to electrical power consumption metrics. The goal of the project is to permit any company or individual to measure the power consumption of its tech services and get this data in a convenient form, sending it through any monitoring or data analysis toolchain.

As part of its integration with OpenNebula, a Scaphandre agent is installed on each hypervisor host, which is in charge of collecting the consumption metrics. Using the Prometheus exporter provided by Scaphandre, the metrics are exported and stored in Prometheus and can be later queried from Grafana. Scaphandre gathers an estimation for the power consumption for each process in a physical machine using the CPU RAPL[8] extensions. A high level view of the configuration is presented in *Figure 1.2*.

[7] https://github.com/hubblo-org/scaphandre
[8] https://hubblo-org.github.io/scaphandre-documentation/compatibility.html#checking-rapl-is-available-on-your-cpu

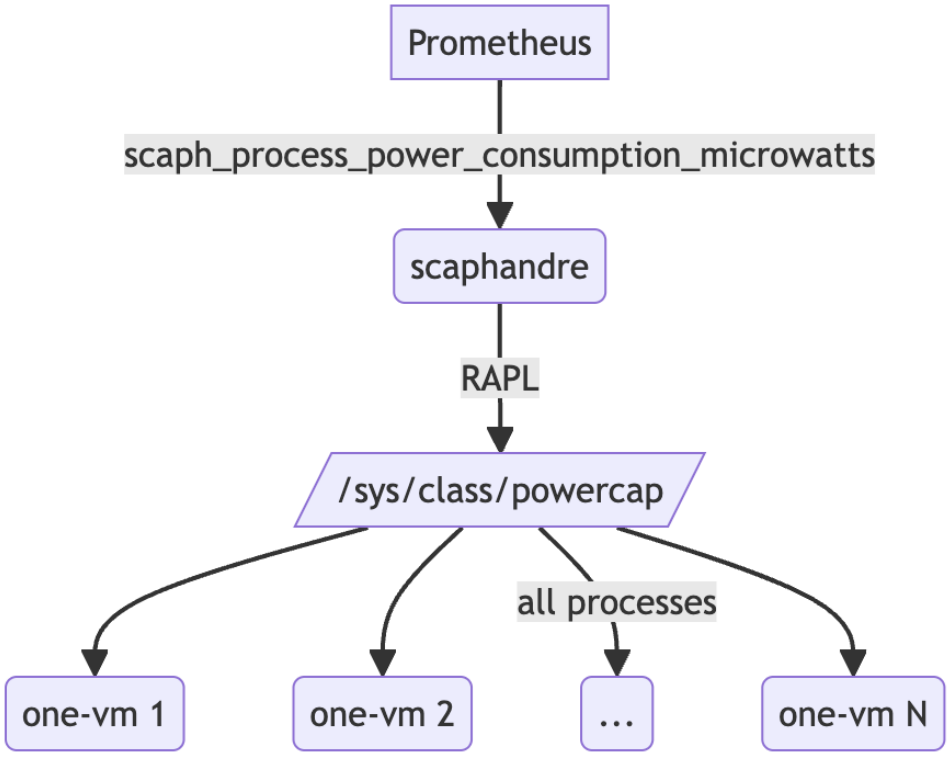

Figure 1.2. OpenNebula Scaphandre Integration enabling VM metric gathering

The power metrics per-VM are computed strictly for each VM. That was accomplished after filtering the power consumption per process reports for the host running every VM via Scaphandre. Scaphandre provides an estimation of the power consumption per each process of the host, accessing to the Linux powercap devices (RAPL extensions) on the physical host. As each VM is a process, power used for each one of them can be isolated.

Intel and AMD processors have been introducing extensions that compute RAPL (Running average power limit) for most of its processors. These extensions account the CPU time as per the frequency that each computing domain uses (being a domain a set of physical resources) and compute a good estimation of the amount of energy that it has been using. The drawback of this approach is the need for relatively new CPUs and operating systems. For instance, in order to access this power capability extensions on Ryzen[9] processors a recent Linux kernel (greater than 6.0) is needed.

Scaphandre correlates the resource usage per each process with the RAPL data, thus allowing us to have a good estimation for the power consumption of each process. Every VM running on a host is an independent process, so this estimation can be considered the power consumption for every VM isolating its Scaphandre metrics for power usage (in microWatts). These measurements reflect only CPU/memory/internal GPU power consumption and are not adding GPU power to it.

[9] https://www.amd.com/en/products/processors/desktops/ryzen.html

Scaphandre offers an integration with Prometheus so the energy metrics are available for consumption to the AI-Enabled Orchestrator through the Prometheus integration.

Geolocation Metrics

A key problem that the COGNIT project aims to give a solution for is delivering low latency function offloading. For this to be optimally solved, metrics on geolocation need to be provided to the AI-Enabled Orchestrator so it can infer latency in the relevant points of the continuum for the particular cloud infrastructure where the COGNIT framework is deployed.

With this extension, every hypervisor host with a public IP address has a GEOLOCATION attribute added to their OpenNebula metadata. This will hold a space separated list of coordinates in the form of `latitude,longitude` corresponding to the geographic location of the public IP address. The open source project geocoder[10] is leveraged to obtain this information.

An OpenNebula hook has been developed that can be tied to changes of state in the OpenNebula hosts. Every time a host enters the MONITORED state, which should be the end result of adding a host, the attribute `GEOLOCATION` should appear in the host template. This effectively implements the integration since there is no need for periodic refreshments of this metric as hypervisor hosts have a static location. This information is available to the AI-Enabled Orchestrator through the OpenNebula API.

[10] https://github.com/alexreisner/geocoder

2. AI-Enabled Orchestrator

The AI-Enabled Orchestrator (AI-O) is the heart of the serverless platform that enables multiple features, including:

- optimal placement of serverless runtimes,
- dynamic migration of applications while changing application requirements, and
- optimal resource utilisation in the cloud-edge continuum proactive migration.

For example, while changing energy usage of an individual cluster according to the energy and sustainability metrics collected by the cloud-edge manager. The AI-O will employ multi-objective optimization to concurrently evaluate the multiple objectives simultaneously from device clients to offloading applications to clusters in the cloud-edge continuum. This will ensure conflicting goals maximising performance while minimising cost, running on green clusters whenever available, and being well-balanced. The main focus of this version of the implementation has been on preliminary research, such as implementing learning models and the end-to-end integration of its subcomponents.

[SR5.1] Building Learning Models

Description

AI-Enabled Orchestrator is key in COGNIT Framework, where AI/ML models are developed to address multiple downstream tasks within the orchestrator, such as characterization and classification of workloads, prediction of workloads, optimization of resource utilisation, proactive migration, and energy-aware placement. The provisioning engine within cloud-edge manager collects both application and resource metrics that integrate with OpenNebula and Prometheus. The AI-Enabled Orchestrator can pull the metrics to train a learning model that can make smart decisions. TimeScaleDB is configured to store the pulled metrics from OpenNebula Prometheus. Consequently, these metrics will be the base for training learning models for diverse downstream orchestrator tasks. Given that monitored metrics lack labelling, unsupervised learning methods are adopted since each model can learn from data without any prior label.

Deep learning (DL) models have been overperforming classical machine learning models in predicting time series data. Although, they are data hungry and expensive in computation. For example, Long Short-Term Memory (LSTM) networks are in the category of recurrent neural networks, which can capture the complex patterns and regularities in the sequence data [1]. Attention mechanisms can help a model to learn the important information in data while discarding unimportant information simultaneously [2]. Transformer models employ attention mechanisms extensively, which can enhance in analysing time series data for different downstream tasks, including prediction [3][4].

As part of AI/ML model development, the AI-O will maintain a model repository for diverse downstream tasks. These models will be selected at runtime according to the downstream tasks. Even though, currently developing number of models for the model repository, AI-O employs two unsupervised learning methods: 1) Multivariate time-series clustering based on common principal component analysis (Mc2PCA) [5]; and 2) Improved deep embedded

clustering (IDEC) [6], which is an enhanced method based on the deep embedded clustering algorithm [7]. The Mc2PCA is a classical learning method, developed inspired by K-means algorithm, whereas the IDEC model is DL-based that integrates the clustering stage into the neural network.

Mc2PCA

Mc2PCA [5] is developed inspired by K-means algorithm. Precisely, this algorithm includes three major steps.

1) First, assign all workload series to N clusters equally ($1 < N < L$), L is the amount of workload series.

2) Secondly, a common projection axis S_k for cluster k ($1 \leqslant k \leqslant N$) is constructed using common principal component analysis (CPCA, a variant of principal component analysis). Equations (1), (2) and (3) illustrate the computation required in CPCA algorithm. Let cluster k contain N time series, calculate covariance matrix C_i for each time series x_i, then get the common covariance matrix $\overline{C}$ by averaging all C_i. Finally, using singular value decomposition (SVD)[11] to decompose $\overline{C}$ and reserve first p components to get the common projection axes S_k ($p = 4$ in the current implementation). Here, the CPCA algorithm applies to all clusters to obtain the corresponding common projection axis.

$$C_i = cov(x_i) = E\left[x_i^T x_i\right] \quad (1)$$

$$\overline{C} = \frac{1}{N}\sum_{i=1}^{N} C_i \quad (2)$$

$$S_k = SVD(\overline{C})(:, :p) \quad (3)$$

3) Thirdly, project and reconstruct virtual machine workload time series x_i to each cluster by corresponding common projection axis S_k (shown in Equation 4), and calculate the error between x_i and reconstructed input x_i' (Equation 5). Then reassign x_i to the cluster k', which has the minimum error. Calculate the overall reconstruction error E after all samples have been reassigned (Equation 6).

$$x_i' = x_i S_k S_k^T \quad (4)$$

$$E_{ik'} = \|x_{ik'}' - x_i\|_2 \quad (5)$$

$$E = \sum_{i=1}^{l} E_{ik'} \quad (6)$$

[11] https://en.wikipedia.org/wiki/Singular_value_decomposition

Step 2 and 3 would be iterated repeatedly until the overall reconstruction error *E* remains unchanged between two adjacent iterations. *Figure 2.1* illustrates the workflow of Mc2PCA algorithm.

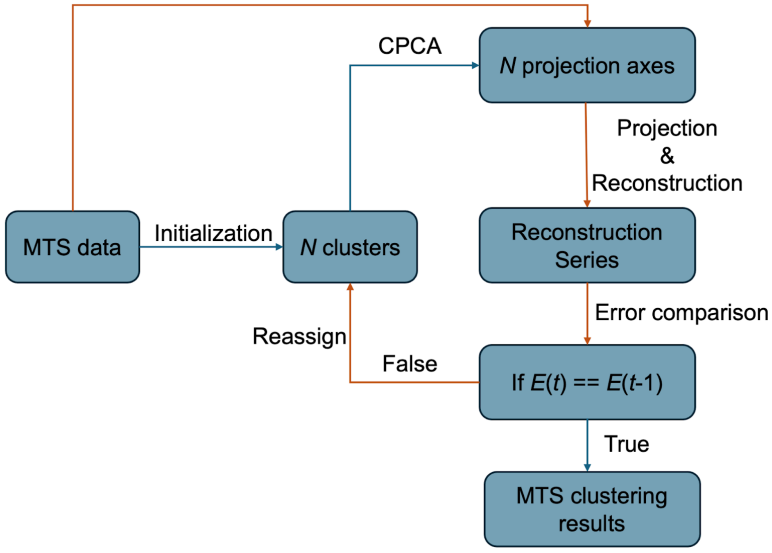

Figure 2.1. The workflow of Mc2PCA algorithm, MTS indicates Multivariate Time Series

IDEC

The IDEC [6] algorithm is a deep learning-based End-to-End clustering algorithm. Specifically, IDEC employs classical Encoder-Decoder architecture to obtain the feature vector of the original workload, i.e., collected metrics as time series, and then carries out clustering based on feature vectors. Figure 2.2 illustrates the architecture of IDEC. The input data of IDEC model is a flattened tensor of multi-variant time series, all network layers in the encoder-decoder model are fully connected layers, while they can also be replaced by another kind of deep learning operator like the 1-D convolution layer.

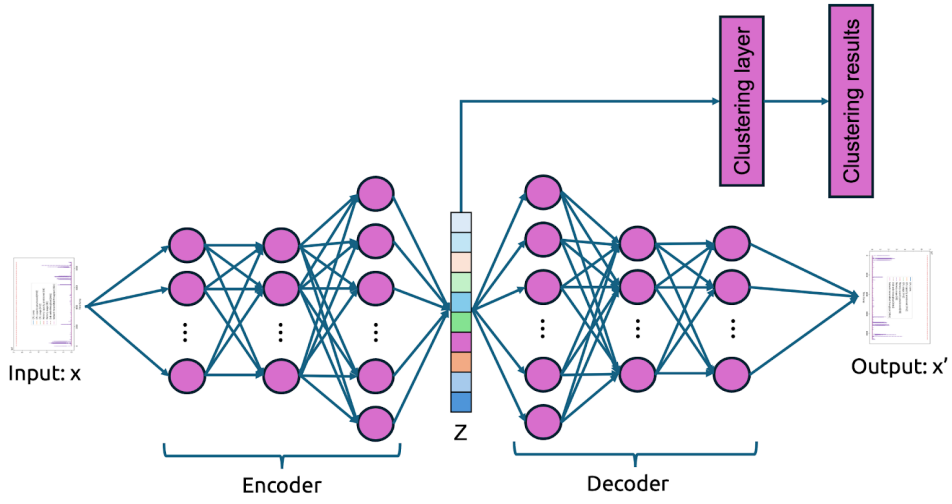

Figure 2.2. IDEC Architecture. z is the feature vector; x is the input workload; x' is the reconstructed input.

The training of IDEC consists of two steps.

1) Train encoder-decoder with single loss of reconstruction measured by Mean Squared Error (MSE):

$$L_r = \sum_{i=1}^{n} \|x - x'\|_2^2 \quad (7)$$

2) Train IDEC model with combined loss for both clustering and reconstruction (L_T).

$$q_{ij} = \frac{(1+\|z_i - u_j\|^2)^{-1}}{\sum_j (1+\|z_i - u_j\|^2)^{-1}} \quad (8)$$

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_j q_{ij}^2 / \sum_i q_{ij}} \quad (9)$$

$$L_c = KL(P\|Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \quad (10)$$

$$L_T = L_r + \gamma L_c \quad (11)$$

z_i is the feature vector extracted by encoder in IDEC model, u_j is the clustering center of cluster k (0 < j < k), which is initialized by k-means algorithm. q_{ij} is the similarity between feature vector z_i and cluster center u_j measured by student's t-distribution. P_{ij} is the auxiliary target distribution. The clustering loss L_c is defined by KL (Kullback-Leibler)-divergence between P and Q. From Equations (8), (9) and (10), it is obvious that P_{ij} is decided by q_{ij} which demonstrates that IDEC is a self-supervised learning algorithm, also an unsupervised learning algorithm. In Equation (11), $\gamma > 0$ is a coefficient that controls the degree of distorting embedded space.

In the inference step, the decoder part is removed, and only the encoder as well as the clustering layer are required.

Data

Both models are ready for evaluation with any of these three different data sources: public or benchmark dataset, testbed dataset, and emulator dataset. Here, the analysis and results are reported based on a public dataset.

Public dataset: The GWA-t-12 Bitbrains dataset was used to validate the inspired models. It contains historical performance data of 1750 virtual machines from a distributed datacenter in TU Delft, which is a service provider that is specialised in managed hosting and business computation for enterprises. The data contains 10 metrics, namely:

1. CPU cores: number of virtual CPU cores provisioned.
2. CPU capacity provisioned (CPU requested): the capacity of the CPUs in terms of MHz, which is equal to number of cores x speed per core.

3. CPU usage: in terms of MHz.
4. CPU usage: in terms of percentage.
5. Memory provisioned (memory requested): the capacity of the memory of the VM in terms of KB.
6. Memory usage: the memory that is actively used in terms of KB.
7. Disk read throughput: in terms of KB/s.
8. Disk write throughput: in terms of KB/s.
9. Network received throughput: in terms of KB/s.
10. Network transmitted throughput: in terms of KB/s.

Testbed data: The primary goal of the project is to deploy the developed models in the COGNIT Framework to validate their performance, precisely for different downstream tasks in the orchestrator. Device client offloads the task to serverless runtimes that needs to be deployed in an edge cluster according to the requirements and availability. Intuitively, cloud-edge manager monitors and collects data as reported in Section 1.

Emulator data: The COGNIT testbed setup lacks diversity of deployed applications which is required to verify the scalability of the model and systems. So an emulator has been developed in parallel. However, the emulator is integrated with the COGNIT testbed. So it is possible to verify the developed models in both scenarios.

Data generation using the emulator

The purpose of the emulator (available here, see *Figure 2.3*) is to replicate the essential functionality of the COGNIT testbed to be able to deploy representative workloads to generate the necessary data for training and validation of the ML models for the AI-O. The emulated data is not exactly representative of the real environment although as it is generated we can easily label it, thus allowing us to quickly validate our models, enabling faster prototyping and debugging. It also allows us to start building models for yet-to-be-implemented functionality, as we can prototype different models on the synthetic data.

The emulator tries to be as close as possible to the testbed while being able to emulate larger systems on less powerful hardware. With this in mind and also searching for a time-effective solution in terms of development, we chose Prometheus for monitoring, as the testbed uses it, thus making it more coherent and the AI-O Connector easier to develop. However, as the emulator SRs are implemented using containers, we decided that using cAdvisor to extract the SR metrics would be the easiest solution.

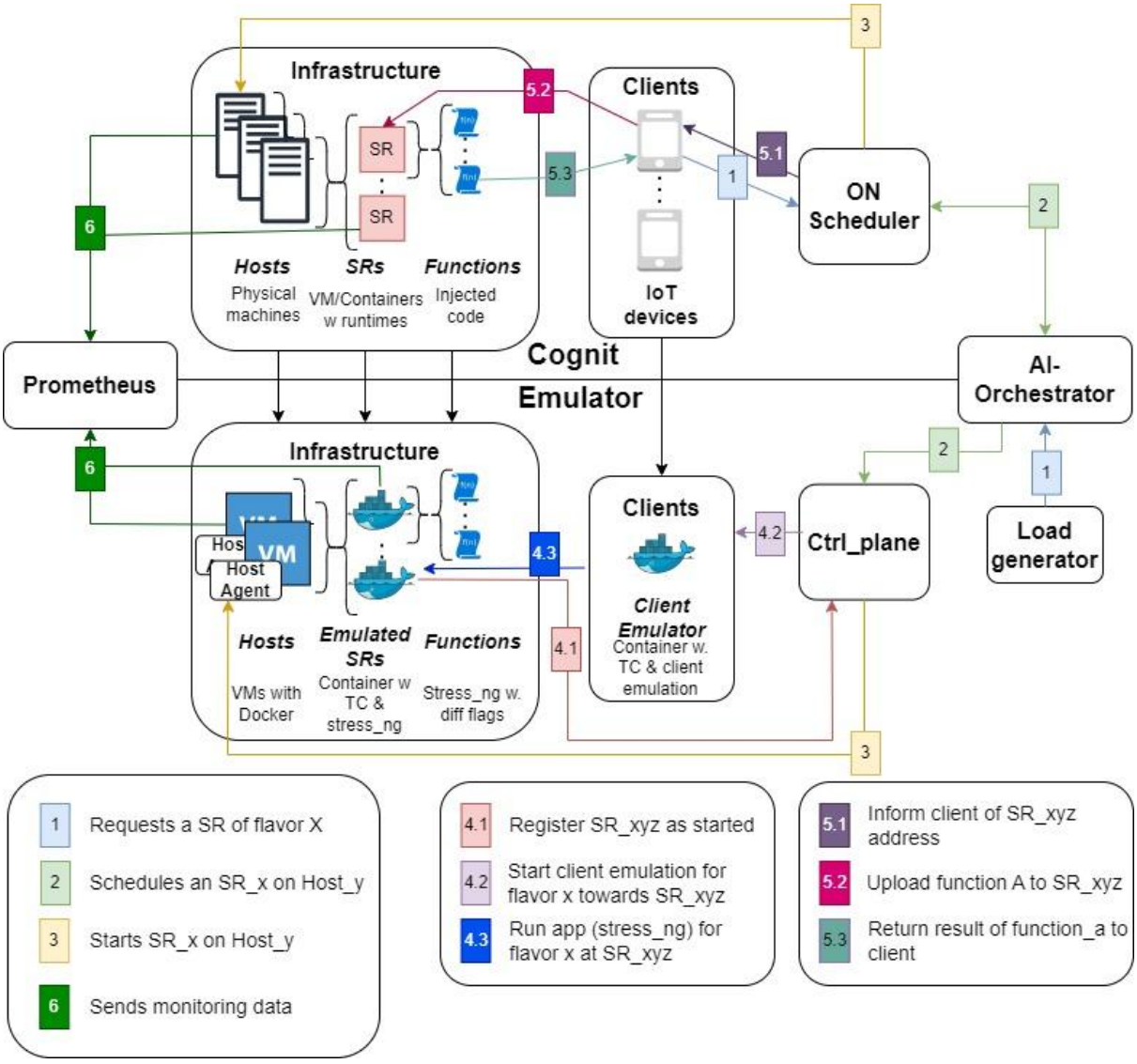

Figure 2.3: AI-O Emulator Architecture

The applications are represented by a set of stress tests from stress-ng[12]. This allows the creation of applications with distinct performance characteristics, simplifying the validation of the workload classifiers. Further validation through developing applications with overlapping performance characteristics thus only requires running multiple stress tests simultaneously. Application requests are generated from the *Client Emulator* container. This container exposes an API to start sending requests to a specified IP (an *Emulated serverless runtimes* IP).

The *Emulated SR* is kept lightweight and generic, with containers providing an API for running stress-ng with custom parameters. The hosts for the *Emulated SRs* are kept as simple Ubuntu VMs with Docker, cAdvisor, and a host agent that exposes a simple API for controlling the containers. To integrate a host one needs to clone the emulator repository

[12] https://wiki.ubuntu.com/Kernel/Reference/stress-ng

to the host, configure the environment variables (see .env), and then run the install_for_hosts.sh script.

The *Ctrl_Plane* emulates the CE-M of Cognit. This component is responsible for managing the *Emulated SRs* through the hosts, as well as initiating new clients on the *Client Emulator*. To allow the AI-O to seamlessly integrate with the emulator it also replicates the interface between the AI-O and the CE-M. This opens up the future possibility of training reinforcement learning agents for the AI-O using the emulator.

We have also partially developed the functionality to modify the network configuration between components to emulate a deployment across a Cloud-Edge environment using Traffic Control (TC). This allows us to emulate more complex network configurations than the COGNIT testbed can provide. However, this functionality is as of yet unfinished.

Results

Both inspired models are implemented based on the public dataset with 7 different metrics, both algorithms were trained and tested with 56049 and 14013 samples, respectively. The algorithms are evaluated and compared from the perspectives of precision and latency. For precision, Silhouette metric is employed since it can quantify how well a sample fits into its assigned cluster and how distinct it is from other clusters, the range of Silhouette Score is from -1 to 1, while a higher score demonstrates superior clustering precision. For latency, we compare the elapsed time of processing 1000 samples, which can be regarded as 1000 virtual machines waiting for scheduled in the real application. The Silhouette Scores of Mc2PCA and IDEC algorithms are 0.029 and 0.92, and latency are 0.026s and 0.195s, respectively. Figures 2.4 to 2.7 illustrate the results of two algorithms, the feature vector's dimensionality has been decreased to 2 by PCA for visualisation.

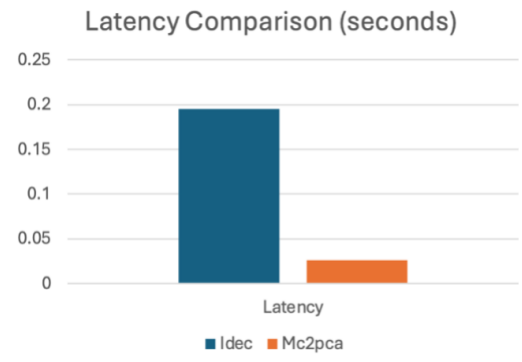

Figure 2.4. IDEC: 0.195s, Mc2PCA: 0.026s.

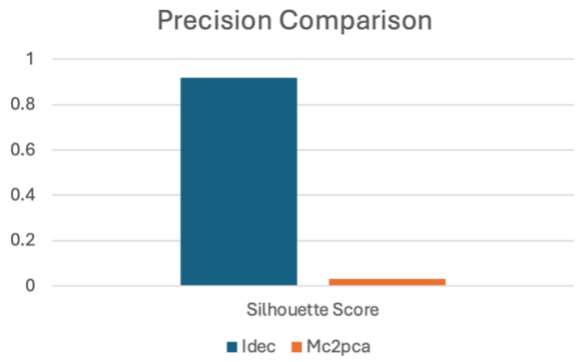

Figure 2.5. IDEC: 0.92, Mc2PCA: 0.029.

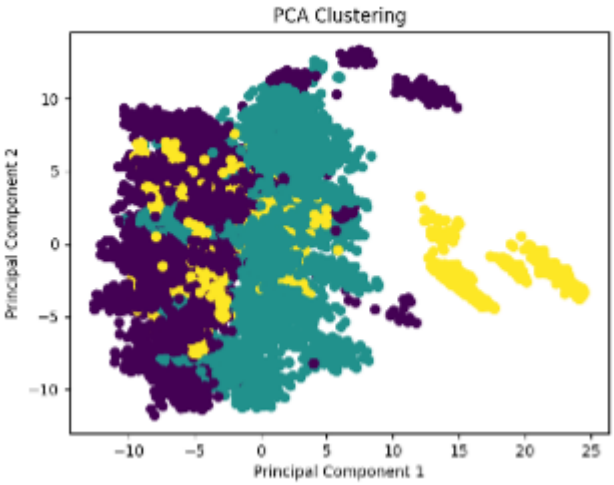

Figure 2.6. Clustering results of Mc2PCA.

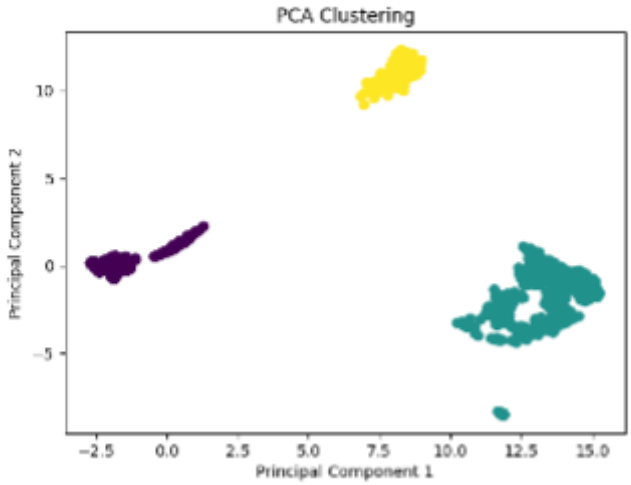

Figure 2.7. Clustering results of IDEC.

From the current implementation and experimental analysis, IDEC model shows higher silhouette score than Mc2PCA, which indicates that the neural network is stronger in extracting features and learning without manual effort. Hence, more advanced DL-based algorithms will be designed and implemented as a follow-up to this and added to the model repository.

[SR5.2] Smart Deployment of Serverless Runtimes

Description

The AI-Enabled Orchestrator is responsible for multiple downstream tasks that include optimal resource utilisation, placement of serverless runtimes, dynamic migration of applications, and proactive migration across infrastructures. This component takes a set of SRs and the corresponding valid hosts and optimises the placement according to the selected scheduling model. This can be achieved by integrating the historical data of the SRs and the host data to a multi-objective optimization that schedules serverless runtimes and aims to optimise against resource contention and green-energy utilisation.

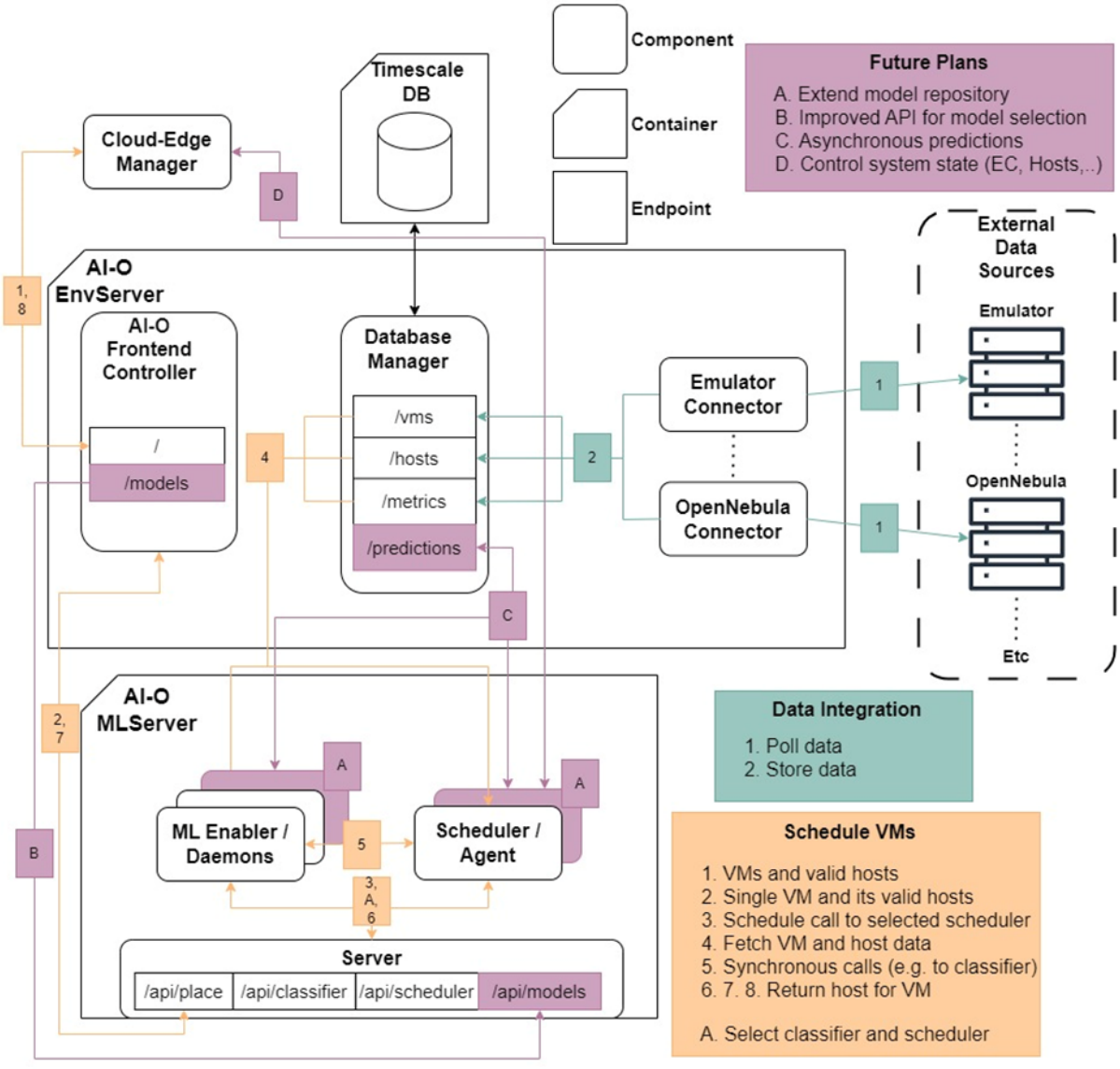

Figure 2.8: Architecture of the AI-Enabled Orchestrator

Architecture & Components

The AI-O is decomposed into multiple subcomponents, including the database, the environment (env) server, and the ML server. Each component consists of multiple parts, see Figure 2.8. This allows for a modular design allowing individual parts to be modified, scaled or swapped completely without impacting the rest of the system. This is particularly important for the machine learning parts as we plan to provide a model repository where the user can choose models depending on their use-case.

The AI-O deploys a local database that caches data from external data sources, which could be Emulator's prometheus service, OpenNebula prometheus, and OpenNebula core oned daemon. This offers the following benefits:

- Locally cached data, decreasing network bandwidth and latency while beneficial in re-training scenarios and fast scheduling decisions.
- Arbitrary schema enables custom metrics, e.g., asynchronous ML-based analysis, or pre-processing at ingestion (the Connector in Figure 2.8).
- Custom retention and/or aggregation policies.

Precisely, there is a cost of duplicating data and potentially fetching unnecessary data from the external data sources . If that cost is substantial, this could be circumvented thanks to the modular design of the AI-O. The architecture abstracts data fetching at the Database Manager (see Figure 2.8), thus, the Connector and Database Manager could be changed to instead fetch certain data synchronously from the external data source. Although it is a relatively small change, it will not be explored in this project.

External data sources can be any arbitrary data source. For this project, we are developing custom connectors for reading data from an Emulator’s Prometheus service, OpenNebula’s Prometheus, and oned services. Integrating against other data sources, e.g., a Kubernetes cluster, or weather metrics, only requires a Connector that needs to be developed.

The ML-Enabler/Daemons are non-decision-making models, primarily focused on data compression/extraction and forecasting. There are two types of models, first, asynchronous, continuously providing the most recent predictions without affecting the decision-making time, particularly useful for slower-executing models. Secondly, we have the synchronous models, lighter-weight models that are either significantly impacted by data staleness or unnecessary to execute at predefined intervals. For the M15 (March 2024) deliverables, we have developed a set of classifiers, these do not make decisions but instead extract a compressed representation of a system's raw time series data, simplifying the scheduling models, see below in Schedule VMs for further information on their integration. Here, a VM may hold multiple serverless runtimes.

The Scheduler/Agent models are the action-performing/state-changing models, focusing on optimising the Cloud-Edge continuum according to specified objectives. There are multiple potential optimization areas, e.g., number and location of Edge Clusters, size and number of hosts, VM placement within or between Edge Cluster, caching and storage, networking, etc. The execution latency on these models might also vary significantly, even with different models tackling the same problem. For example, initial VM placement

requests should be resolved quickly whereas optimising the VM placement through migrations is usually less time-sensitive. Therefore, the models are separated, simplifying the training and architecture of each individual model. For example, in M15 we developed an interference-aware scheduler, using the VM representation from the above-mentioned classifier, to intelligently place VMs across the edge clusters.

Schedule VMs

Scheduling VMs using the AI-O is an eight-step process (shown in orange in *Figure 2.8*).

1. AI-O frontend receives the request for a set of SRs (serverless runtimes) and their corresponding valid hosts (see Deliverable D4.1 further info).
2. Frontend forwards these as separate scheduling requests to the ML Server.
3. ML Server forwards the request to the currently selected scheduler and classifier.
4. Scheduler and classifier fetch historical data about the SRs and hosts from the Database Manager.
5. Scheduler makes a scheduling decision based on the state of the hosts and the classification of the SR workload characteristics (using the currently selected classifier).
6. Returning the scheduling decision from scheduler to ML server.
7. Returning the scheduling decision from ML server to AI-O frontend.
8. AI-O frontend waits for all SRs to be scheduled and lastly returns the scheduling decisions to its caller.

Database Manager APIs

The database manager has three API endpoints: */vms*, */hosts*, */metrics* (shown in Table 2.1, *Figures 2.9 - 2.15*). Each of these allows retrieval and storage of corresponding data. The VMs and hosts tables consist of the latest metrics and state for each object. VMs have: id, state id, deployed status, deployed host id, CPU count and usage, available and used memory, disk read and writes, and finally network reads and transmits. Hosts have: id, CPU count and usage, available and used memory, disk read and writes, network reads and transmits, and finally energy usage. Metrics are time series formatted, having: id, object type (VM or host), CPU usage, memory usage, disk read and write, network reads and transmits, and finally energy usage.

Action	Verb	Endpoint	Request Body	Response
Get all the VMs, states and most recent metrics.	GET	/vms	A JSON formatted according to *Figure 2.9*.	Status code 200 (Success) if the execution was successful. An array version of JSON formatted according to *Figure 2.11*. 400 (Bad request) if there is any error.

Get the VM's state and most recent metrics (VMid specified in the URL).	GET	/vms/:id	A JSON formatted according to *Figure 2.9*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.11*. 400 (Bad request) if there is any error.
Add a VM and its state and initial metrics.	POST	/vms	A JSON formatted according to *Figure 2.11*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.9*. 400 (Bad request) if there is any error.
Delete the VM (VMid specified in the URL). Note: Does not remove its metrics.	DELETE	/vms/:id	A JSON formatted according to *Figure 2.9*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.9*. 400 (Bad request) if there is any error.
Update the VM's host binding (VMid and hostid specified in the URL).	PUT	/vms/:id/:hostid	A JSON formatted according to *Figure 2.9*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.10*. 400 (Bad request) if there is any error.
Delete the VM's host binding (VMid and hostid specified in the URL).	DELETE	/vms/:id/:hostid	A JSON formatted according to Figure 2.9.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.10*. 400 (Bad request) if there is any error.
Get all the Hosts, states and most recent metrics.	GET	/hosts	A JSON formatted according to *Figure 2.9*.	Status code 200 (Success) if the execution was successful. An array version of JSON formatted according to *Figure 2.12*. 400 (Bad request) if there is any error.
Get the Host's state and most recent metrics (Host id specified in the URL).	GET	/hosts/:id	A JSON formatted according to *Figure 2.9*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.12*. 400 (Bad request) if there is any error.

Add a Host and its state and initial metrics.	POST	/hosts	A JSON formatted according to *Figure 2.12*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.9*. 400 (Bad request) if there is any error.
Delete the Host (Host id specified in the URL). Note: Does not remove its metrics.	DELETE	/hosts/:id	A JSON formatted according to *Figure 2.9*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.9*. 400 (Bad request) if there is any error.
Get the metrics of an object (VM or Host).	GET	/metrics	A JSON formatted according to *Figure 2.13*.	Status code 200 (Success) if the execution was successful. An array version of JSON formatted according to *Figure 2.14*. 400 (Bad request) if there is any error.
Add metrics to an object (updates the object's most recent metrics as well)	POST	/metrics	A JSON formatted according to *Figure 2.15*.	Status code 200 (Success) if the execution was successful. A JSON formatted according to *Figure 2.9*. 400 (Bad request) if there is any error.

Table 2.1: API that defines the endpoints of the Database Manager

```
Unset
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": { },
}
```

Figure 2.9: JSON Schema for fetching/removing objects.

```
Unset
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "VMID": { "type": "string" },
    "HostID": { "type": "string" },
```

```
  },
  "required": ["VMID", "HostID"]
}
```

Figure 2.10: JSON Schema for setting scheduler and classifier on MLServer.

Unset

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "vmid": {
      "type": "string"
    },
    "stateid": {
      "type": "integer"
    },
    "deployed": {
      "type": "boolean"
    },
    "hostid": {
      "type": "string"
    },
    "hoststateid": {
      "type": "integer"
    },
    "total_cpu": {
      "type": "number"
    },
    "total_memory": {
      "type": "number"
    },
    "usage_cpu": {
      "type": "number"
    },
    "usage_memory": {
      "type": "number"
    },
    "disk_read": {
      "type": "number"
    },
    "disk_write": {
      "type": "number"
    },
    "netrx": {
      "type": "number"
    },
    "nettx": {
      "type": "number"
    }
  },
  "required": ["vmid", "stateid", "deployed", "hostid", "hoststateid",
"total_cpu", "total_memory", "usage_cpu", "usage_memory", "disk_read",
"disk_write", "netrx", "nettx"],
}
```

Figure 2.11: JSON Schema for VM information.

Unset

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "hostid": {
      "type": "string"
    },
    "stateid": {
      "type": "integer"
    },
    "total_cpu": {
      "type": "number"
    },
    "total_memory": {
      "type": "number"
    },
    "usage_cpu": {
      "type": "number"
    },
    "usage_memory": {
      "type": "number"
    },
    "disk_read": {
      "type": "number"
    },
    "disk_write": {
      "type": "number"
    },
    "netrx": {
      "type": "number"
    },
    "nettx": {
      "type": "number"
    },
    "vms": {
      "type": "integer"
    },
    "energy_usage": {
      "type": "number"
    }
  },
  "required": ["hostid", "stateid", "total_cpu", "total_memory",
"usage_cpu", "usage_memory", "disk_read", "disk_write", "netrx", "nettx",
"vms", "energy_usage"],
}
```

Figure 2.12: JSON Schema for Host information.

Unset

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "hostid": {
      "type": "string"
    },
    "metrictype": {
```

```
      "type": "integer"
      "description": "0 for host, 1 for VM"
    },
    "since": {
      "type": "integer"
      "description": "time to fetch from then until now in nanoUnixTime"
    },
    "count": {
      "type": "integer"
      "description": "max number of entries to fetch"
    }
  },
  "required": ["hostid", "metrictype", "since", "count"]
}
```

Figure 2.13: JSON Schema for fetching metrics.

Unset

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Metric",
  "type": "object",
  "properties": {
    "timestamp": {
      "type": "string",
      "format": "date-time"
    },
    "cpu": {
      "type": "number"
    },
    "memory": {
      "type": "number"
    },
    "disk_read": {
      "type": "number"
    },
    "disk_write": {
      "type": "number"
    },
    "netrx": {
      "type": "number"
    },
    "nettx": {
      "type": "number"
    },
    "energy_usage": {
      "type": "number"
    }
  },
  "required": ["timestamp", "cpu", "memory", "disk_read", "disk_write",
"netrx", "nettx", "energy_usage"],
}
```

Figure 2.14: JSON Schema for metrics.

```
Unset
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "id": {
      "type": "string"
    },
    "metrictype": {
      "type": "integer"
      "description": "0 for host, 1 for VM"
    },
    "timestamp": {
      "type": "string",
      "format": "date-time"
    },
    "cpu": {
      "type": "number"
    },
    "memory": {
      "type": "number"
    },
    "disk_read": {
      "type": "number"
    },
    "disk_write": {
      "type": "number"
    },
    "netrx": {
      "type": "number"
    },
    "nettx": {
      "type": "number"
    },
    "energy_usage": {
      "type": "number"
    }
  },
  "required": ["id", "metrictype","timestamp", "cpu", "memory", "disk_read",
"disk_write", "netrx", "nettx", "energy_usage"]
}
```

Figure 2.15: JSON Schema for adding metrics.

AI-O Frontend Controller

The JSON schema of the data sent and returned for communication between AI-Enabled Orchestrator and cloud-edge manager through an endpoint with REST API remained similar as reported in *Deliverable D4.1*.

AI-O ML Server

The following examples describe JSON schemas (*Figure 2.16-2.18*) and attributes (shown in Table 2.2) of the data sent to the MLServer */place* API and the expected return, respectively:

Action	Verb	Endpoint	Request Body	Response
Request a placement plan for a PENDING Serverless Runtimes	POST	/api/place	JSON representation of the VM associated with pending Serverless Runtime services and the list of viable HOSTs.	Status code 200 (Success) if the execution was successful. A JSON with information about the placement is returned. 400 (Bad request) if there is any error.
Set the scheduling model for placement requests.	POST	/api/scheduler	JSON with model name.	Status code 200 (Success) if the execution was successful. A JSON with information about the placement is returned. 400 (Bad request) if there is any error.
Set the classifier model for estimating VM resource usage.	POST	/api/classifier	JSON with model name.	Status code 200 (Success) if the execution was successful. A JSON with information about the placement is returned. 400 (Bad request) if there is any error.

Table 2.2: Attributes between ML server and AI-O communication

Unset

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
      "VM_ATTRIBUTES": {
        "type": "object",
        "properties": {
          "GNAME": { "type": "string" },
          "UNAME": { "type": "string" }
         },
         "required": ["GNAME", "UNAME"]
      },
      "CAPACITY": {
        "type": "object",
        "properties": {
          "CPU": { "type": "number" },
          "DISK_SIZE": { "type": "number" },
          "MEMORY": { "type": "number" }
        },
        "required": ["CPU", "DISK_SIZE", "MEMORY"]
      },
      "HOST_IDS": {
        "type": "array",
        "items": { "type": "integer" }
      },
      "ID": { "type": "integer" },
```

```
      "STATE": { "type": "string" }
  },
  "required": ["HOST_IDS", "ID"]
}
```

Figure 2.16: JSON Schema for AI-O Frontend to ML Server communication

```
Unset
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "ID": { "type": "integer" },
    "HOST_ID": { "type": "integer" }
  },
  "required": ["ID", "HOST_ID"]
}
```

Figure 2.17: JSON Schema for MLServer to AI-O Frontend communication

The following examples describe a JSON schema of the data sent to the MLServer */classifier* and */schedule* API, including descriptions of the different classifiers and schedulers available. Also, Table 2.3 describes the models for the workload characterization and Table 2.4 indicates the models for the scheduler, respectively.

```
Unset
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "model_name": { "type": "string" },
  },
  "required": ["model_name"]
}
```

Figure 2.18: JSON Schema for setting scheduler and classifier on MLServer.

Name	Description
RandomClassifier	Randomly generates the V_{r_j} for the VM.
DLIR	Calculates the V_{r_j} from the intermediate representation of the AutoEncoder trained for IDEC.
DLClassifier	Calculates the V_{r_j} from the distance to classes using IDEC.
ClassicalClassifier	Calculates the V_{r_j} from the distance to classes using MC2PCA.

Table 2.3: Models for the workload characterization

Name	Description
RandomScheduler	Schedules the VM on a random host in the valid host list.
InteferenceAwareScheduler	Schedules VM using Interference-Aware Scheduling. Selecting the optimal host by minimizing D_{max} through using the $D_{green_{h_i}}$ distance metrics.

Table 2.4: Models for the scheduler

Formal models for an energy-aware continuum systems

At present few formal models of federated Cloud–Edge systems exist—and none adequately represent and integrate energy considerations (e.g., multiple providers, renewable energy sources, pricing, and the need to balance consumption over large-areas with other non-Cloud consumers, etc.)

Energy-aware task migration may initially appear to be a straightforward process, but in production environments it can become extremely complex; effective placement requires intelligent decision-making while taking into account multiple factors including energy providers, energy policies, energy pricing, resource availability, SLO arbitration, etc. This is further exacerbated by the dynamic nature of Cloud–Edge environments, which are highly dynamic, mobile and complex, and above all seen as critical infrastructure that should not suffer from serious disruption.

It is therefore vital that new algorithms, mechanisms and methods to improve energy utilisation in the Cloud Continuum are grounded on formal scientific models that identify and support the huge range of providers, heterogeneous components, interactions, stochastic properties, (potentially contradictory) service-level agreements, pricings, and contractual requirements present in both energy and Cloud–Edge systems.

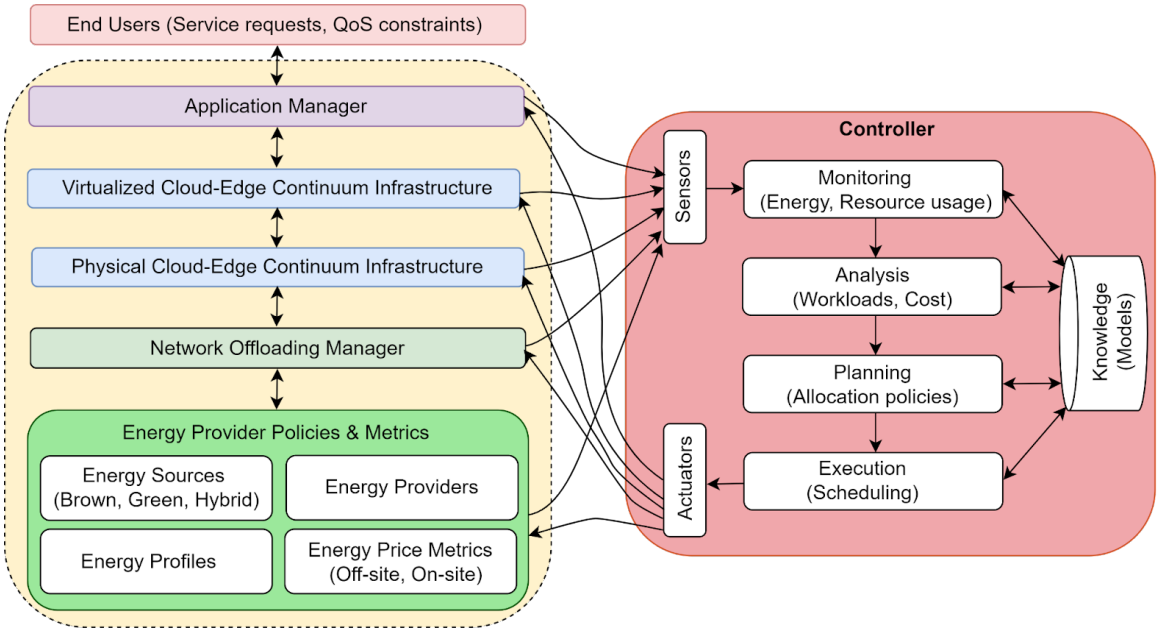

Figure 2.19: A high-level model for energy-aware Continuum systems

Through COGNIT, we have developed an initial high-level model for task placement in a Continuum context, published in [8]. In this work, we propose a perspective model for energy-aware Continuum systems as shown above in *Figure 2.19*.

This has served as one of the foundations for our work into interference-aware scheduling; we plan to develop the model and work further going forward. As a further example, we have developed a sequence diagram to illustrate workload execution for six energy scenarios, shown in *Figure 2.20*.

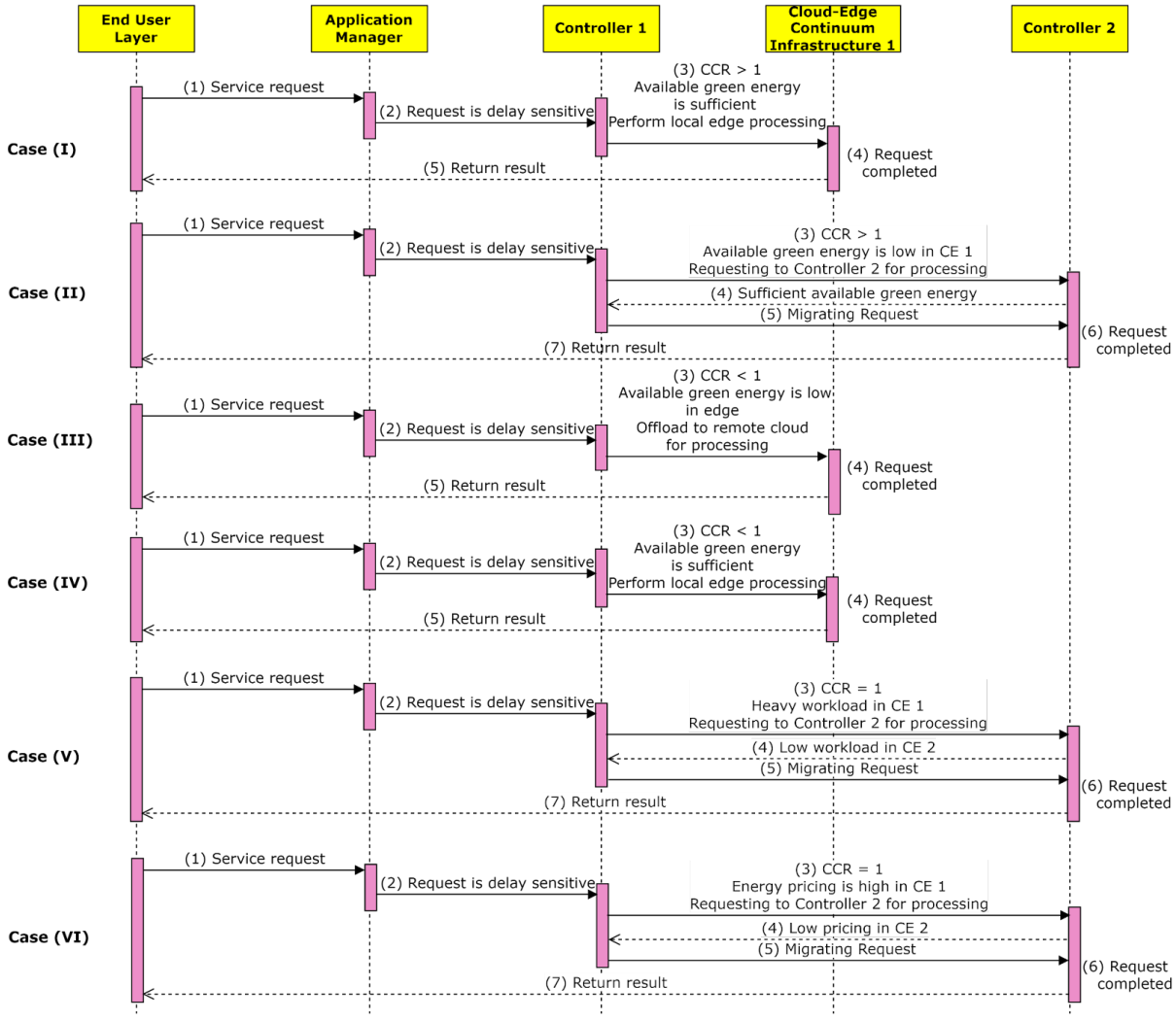

Figure 2.20: Sequence diagram for potential workload placement in energy scenarios

Interference-Aware Scheduling

The first version of the AI-O employs interference-aware scheduling, which has been shown to decrease energy consumption without compromising application performance [7]. The objective of interference-aware scheduling is to minimise contention of hardware resources on the hosts. We carry out this by trying to equalise the resource consumption of the individual hardware resources among all hosts, in turn distributing similar workloads

among the hosts. In principle, picking the host where adding the VM's resource usage would minimise the distance between the hosts' resource utilisation.

Interference-Aware Scheduling:

- Let $H = \{h_1, h_2, ..., h_n\}$ be the set of hosts.
- Let $R = \{r_1, r_2, ..., r_m\}$ be the set of resources (CPU, memory, etc).
- Let $U_{h_i r_j}$ be the utilisation of resource r_j at host h_i.
- Let V_{r_j} be the resource demand of the VM for resource r_j.
- Let D_{h_i} be the distance measure for host h_i .

We are trying to minimise:

$$D_{max} = max(D_{h_1}, D_{h_2}, ..., D_{h_n}) \quad (12)$$

To calculate D_{h_i} we use:

$$D_{h_i} = \sum_{j=1}^{m} \left| \left(U_{h_i r_j} + V_{r_j} \right) - \overline{U_{r_j}} \right| \quad (13)$$

where $\overline{U_{r_j}}$ denotes the average utilisation of resource r_j across all hosts.

To calculate D_{h_i} we need V_{r_j}, this requires an understanding of the resource consumption of each VM. However, having the user specify the utilisation manually is inconvenient and potentially error-prone. Thus, we are instead inferring it from the historical resource usage of the VMs. We employ the workload characterization (see Section SR5.1) that provides the foundation of each class of workloads and representation. For example, calculating a VM's distance to each of the classes gives us a proxy for its resource utilisation, assuming the classes represent distinct types of resource utilisation (e.g., CPU, memory, networks). If we have three classes (c_1, c_2, c_3) for the classifier we get that $R = \{d(c_1), d(c_2), d(c_3)\}$, where the function d measures the distance to the center of the cluster.

Similarly, the intermediate representation of the Auto-Encoder (AE) should represent the resource utilisation of the VM. These are not directly representative of the typical resource metrics, such as CPU and memory, but we are exploring methods to correlate them to these metrics. For example, by looking at the change in resource utilisation at the host when deploying VMs with known vectors one can estimate the effect of each vector index relative to the host resources, thus giving us scalars from VM vectors to traditional metrics such as CPU. If we have the traditional metrics, $R_{trad} = \{r_1, r_2, ..., r_m\}$ and the classifier metrics, $R_{class} = \{d(c_1), d(c_2), d(c_3)\}$ then we plan to be able to estimate the scalars $S_i = \{s_{1r_i}, s_{2r_i}, s_{3r_i}\}$ to go from R_{class} to R_{trad} by $r_i = s_{1r_i} * c_1 + s_{2r_i} * c_2 + s_{3r_i} * c_3$.

As this reverse lookup of metrics has yet to be developed our initial version of the Interference-Aware Scheduler does not schedule based on traditional metrics, e.g., CPU. Instead, it balances the host workload by evenly scheduling either the different cluster

classes, or the vector from the intermediate representation of the AE. However, this requires historical data on the VM's resource usage. To circumvent this the VMs are tagged with a flavour, allowing us to estimate the typical/average consumption of that flavour and can thus be used for initial placement. Flavours without historical data can either be estimated using the most common VM resource usage, a randomised resource profile, or an average resource profile. Then, once the VM/flavour has been properly classified it can be rescheduled accordingly.

This approach is easily extended to consider further optimization criteria. For example, the first iteration of the scheduler includes a green-energy optimizer. Scaling a VM's resource usage relative to a host's green-energy availability will prioritise deploying on the hosts provided with more green energy. This gives us an easily tunable parameter for optimising green-energy utilisation in the system. Similarly, though not yet developed, we could for example penalise deployments on over-allocated hosts, particularly important for heterogeneous environments.

Green-energy adjustment of D_{h_i}:

- Let $g(h_i)$ give the green energy percentage of the host
- Let s_{green} be the green-energy scalar

$$D_{green_{h_i}} = D_{h_i} + D_{h_i} * \left(1 - g\left(h_i\right)\right) * s_{green} \quad (14)$$

This could be extended to any arbitrary cost function F as such, allowing for smarter scheduling in the future:

$$D_{green_{h_i}} = D_{h_i} + F(g_{h_i}, D_{h_i}, \ldots) \quad (15)$$

[SR5.3] Scheduling Mechanisms

It became clear in this cycle that COGNIT needs a way to notify the AI-Enabled Orchestrator that a particular Serverless Runtime has been updated, which can be achieved thanks to the work performed in SR3.1 (see Deliverable D3.2).

The mechanism used to implement this notification is the OpenNebula reschedule functionality. When a Virtual Machine is in the running or power-off state, it can be rescheduled to a different host by enabling (through the API) a rescheduling action over this particular VM. In the next scheduling interval, the VM will be considered by the OpenNebula scheduler for a rescheduling action, and thanks to the work in SR5.3 presented in D4.1, all the VMs subject to be scheduled or rescheduled are delegated to the AI-Enabled Orchestrator.

This SR has been improved in the context of the Provisioning Engine development (SR3.1), since the reschedule action is automatically triggered by the Provisioning Engine on any update of a Serverless Runtime, that potentially can change the VM characteristics in a way the merits a migration of hypervisor for the SR.

3. Energy-efficiency optimization and sustainability

The availability of renewable energy is in many locations intermittent and varies based on the regional or local energy mix, time of day, and season. For some renewable energy sources, such as solar and in particular wind, the supply could fluctuate significantly throughout the day and be hard to predict. Scheduling workloads based on availability of renewable energy in a continuum, or a cloud-edge federation, therefore requires some degree of flexibility in either where the workload can be processed, or when.

In the cloud-edge continuum, the provisioning of compute resources in different locations can be planned ahead of time in anticipation of renewable energy availability, to increase the share of renewable energy. The cloud-edge continuum therefore presents both opportunities and challenges for integrating renewable energy sources to ensure low cost and sustainability.

Challenges for integrating and increasing the share of renewable energy in the cloud-edge continuum

One of the *first challenges* is knowing whether a specific cloud or edge cluster is using green energy. Current approaches usually consider the energy mix in the regional power grid over a time period, and apply carbon accounting methods based on that. There are services (e.g., model as a service) for predicting and estimating renewable energy supply in advance that could be used for capacity planning. However, the situation is more complicated when taking local microgrids and energy storage into consideration - these are very different from the national power grid.

It is common to predict the availability of renewable energy day-ahead (24 hours prediction), which can be updated throughout the day. Given such a prediction for the availability of renewable energy in different locations, the next challenge is planning the compute capacity in different cloud or edge clusters to match the patterns of renewable energy supply. This involves predicting the future workloads, or computing demand, for the next period e.g. next 24 hours, to plan the infrastructure capacity. Since workloads differ in terms of requirements, it is important to take this into consideration, balancing for example latency requirements and other location-based policies or restrictions, with energy-efficiency and renewable energy utilisation.

Grid congestion, i.e., when electricity demand exceeds the capacity of the grid, is another challenging issue - for example the electricity could have been used for other purposes (such as producing green steel).

Predicting resource demand in the cloud-edge continuum

From the perspective of the AI-Enabled Orchestrator, one of the major challenges is to predict future compute resource demands to plan for infrastructural needs, such as provisioning new cloud or edge resources. This requires balancing requirements such as low latency and location-based policies with energy efficiency and demand for renewable energy utilisation. One solution to this problem is to collect historical data of computational resource usage and detect patterns such as peak load, and then let users

set a threshold for how much the system should be over-provisioned. However, over-provisioning may unfortunately decrease the energy efficiency as well as result in increased costs. On the other hand, allocating too few resources may result in an SLA violation or poor performance [9].

To find the right balance and add flexibility, users could be allowed to define boundaries on how much they are willing to compromise on SLA requirements to prioritise renewable energy usage. For instance, a developer could set a limit that a particular application could tolerate up to 100 ms latency and then set a rule to prioritise renewable energy. In this scenario, an edge cluster running on renewable energy would be chosen if it meets the latency threshold of 100 ms, even if there is another non-renewable powered edge cluster offering lower latency.

Another option could be to set a limit and boundaries on how long a particular function call can be delayed. In some scenarios, such as when offloading an emergency response function to detect wildfires, it is crucial that the computation is executed promptly. However, in other cases, such as federated learning or fine-tuning machine learning models, computations could perhaps be postponed until they can be executed on a cluster powered by renewable energy.

One challenge is that provisioning new clusters may take time, which means that the AI-O needs to accurately predict the total computational demand for the upcoming period, such as the next hours, and also recommend when certain clusters need to be decommissioned. Automatically scaling and provisioning cloud resources has been extensively studied in previous work, e.g., for time-sensitive loads [10], carbon-aware day-ahead virtual capacity planning using forecasts of inflexible and flexible loads and load shaping [15, 16]. However, in the context of a cloud-edge continuum to provision new edge clusters remains an open challenge, in particular considering balancing renewable energy and SLAs.

Energy-efficient resource management and scheduling

Once the virtual infrastructure has been provisioned, incoming workloads should be scheduled for processing at the different cloud/edge locations and hosts. This problem operates on smaller time scales than the planning of virtual compute capacity. There are a number of things to take into consideration:

- The power use of a server scales according to its CPU utilisation, as a quadratic function of its frequency, and is often around 30 % of maximum power in idle mode. For most cloud servers, the sweet spot for performance/power ratio is at around 70 % to 80 % CPU utilisation [11].
- On the other hand, the power needed for cooling goes up with server loads/utilisation, and depends non-linearly on the power used by the server. It is generally much more difficult to model.

Holistic resource management is an approach to schedule workloads that improve energy-efficiency from a system perspective, without sacrificing Quality of Service (QoS) and Service Level Agreements (SLAs). That is, to place workloads to jointly optimise both 1) the performance/power ratio of the server equipment, and 2) the energy used by the

cooling system. In holistic resource management, the aim is therefore to consolidate and move workloads between hosts to optimise the overall energy usage of a data center. The objective function for the optimization may include terms for total energy use, renewable energy utilisation, QoS measures, and SLA violations.

In [12], they formulate an optimization problem with the objective to minimise the amount of non-renewable energy usage, assuming that workloads belong to one of two categories: *interactive* (latency-critical) and *batch* (that can be deferred to a later time). To match renewable energy supply with server loads, they defer batch jobs to times of higher renewable energy supply.

The approach of [11] uses a Gated Graph Convolution Network to score candidate scheduling decisions based on three inputs: 1) the candidate matching between workloads and hosts, 2) task dependencies (they modelled a job as a collection of tasks), and 3) host thermal characteristics. The score is a weighted sum of average normalised energy consumption, average normalised temperature and ratio of job SLA violations. To reduce computational complexity, the scheduling decision only considers the K most power hungry tasks and the K most energy-efficient hosts at each scheduling iteration. In a subsequent work, the Gated Graph Convolution Network was replaced with a Convolutional Neural Network [13].

In [17], the utilisation rate of the servers is jointly optimised with the control of the cooling system to minimise energy usage, using reinforcement learning. The inputs were server load, server outlet temperatures, and the duration and load of the different jobs. The reward function considered the power usage of the cooling system, the ratio of dropped jobs, and a penalty for exceeding a cold aisle temperature of 27 degrees Celsius.

The operation of an edge node is optimised in a microgrid setting with onsite renewable energy generation and a battery storage for electricity in [14], taking into account the battery state of charge and renewable energy supply forecasts.

These approaches either assume that the temperatures can be measured, or estimated using a simplified physical model of the system, and that the energy used for the cooling system can be directly calculated from these temperature measurements or estimates. Energy modelling and optimization in the cloud-edge continuum is highly challenging since:

- The hardware, server configurations, and cooling system characteristics are highly heterogeneous in the cloud-edge continuum. The energy models need to be adapted to each cloud or edge site.
- The cooling systems are operated by the infrastructure providers. In a federated cloud-edge continuum, the meta-orchestrator making the scheduling decisions will not be able to directly control the cooling system.

4. Conclusions and future work

The Cloud-Edge Manager and AI-Enabled Orchestrator components have undergone significant development since the M9 (September 2023) implementation. This report describes the components in detail, and discusses the changes and improvements that have been made. Specifically, the Cloud-Edge Manager now deploys Serverless Runtimes across a real federated environment (testbeds located at RISE in Sweden and OpenNebula in Spain), and has been augmented with improved mechanisms to report service readiness. Additionally, the Cloud-Edge Manager now has the ability to report two new classes of metrics that are seen as essential for intelligent orchestration in the Cloud Continuum: geo-location metrics (to assist in delivering low-latency) and virtual machine energy metrics (to improve energy efficiency and use of renewable resources across the Continuum).

This work feeds into the AI-Enabled Orchestrator component, which subsequent to M9 is now fully integrated with the Cloud-Edge Manager. In part using the newly available energy and geo-location metrics, significant work has been performed on unsupervised learning methods - seen as essential to provide predictive analytics in environments that lack labelled data. A model repository has been developed, and two unsupervised learning methods have been implemented: MC2PCA and IDEC; this report describes these methods in detail. Importantly, the ability to reschedule Serverless Runtimes is now available within the Orchestrator - this enables the movement and continued orchestration of existing runtimes, rather than the static deployments that existed in M9. Finally, to improve the reasoning capability of COGNIT, work has been performed and published to develop formal models for an energy-aware Cloud Continuum; this will form the basis going forward for more performant and efficient deployment strategies with a particular focus on energy. Currently this has informed our work on interference-aware scheduling.

To reduce the energy use of, and the amount of non-renewable energy used by, the cloud-edge continuum, it is important to predict compute demand and how it can be scheduled over time and across locations, and proactively adapt the compute capacity accordingly. Measures need to be implemented to:

- Identify the most energy efficient of the participating edge clusters (possibly depending on the specific application/workload).
- Use forecasts of local availability (and carbon intensity) of renewable energy (both from the grid and on-site), to produce forecasts of available compute capacity running on renewable energy, across locations. Both grid and on-site energy supply and energy storage solutions need to be considered and optimised.
- Forecast the compute demands and dependencies of applications, across time and space.
- Proactively schedule the workloads and infrastructure to pair workloads with the right hosts, to improve system-level energy performance.
- More research is needed to understand how data centers can participate in the local energy markets to reduce grid congestion and improve overall grid resiliency

and reduce system-wide carbon emissions, including monitoring of, and participation in, smart grid initiatives and standardisation.

In summary, to enable grid-aware computing, and promote data centers as active participants in future grid services, a holistic approach is needed to better adapt the current datacenter energy systems to developing standard energy market frameworks. It is necessary to investigate which grid services that datacenters should target, and develop interoperability frameworks for the multi-provider cloud-edge continuum based on local regulations and energy mix, as well as integrating their current Energy Management Systems (EMS) with local flexibility market operators and service platforms. Last, but not least, a sustainability assessment framework and methodology is needed to critically evaluate the above, given the local dynamics of carbon intensity and energy mix.

The COGNIT Project aims to contribute to all these points in the coming research and innovation cycles, with a special focus on the challenges associated with energy efficiency optimization and smart placement by enabling AI/ML models across the cloud-edge continuum.

References

[1] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*, 1735-1780.

[2] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR, abs/1409.0473*.

[3] Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. *Neural Information Processing Systems*.

[4] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2020). Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. *AAAI Conference on Artificial Intelligence*.

[5] Li, H. (2019). Multivariate time series clustering based on common principal component analysis. *Neurocomputing*, 349, 239-247.

[6] Guo, X., Gao, L., Liu, X., & Yin, J. (2017). Improved Deep Embedded Clustering with Local Structure Preservation. *International Joint Conference on Artificial Intelligence*.

[7] Mendoza, D., Romero, F., Li, Q., Yadwadkar, N. J., & Kozyrakis, C. (2021, April). Interference-aware scheduling for inference serving. In *Proceedings of the 1st Workshop on Machine Learning and Systems* (pp. 80-88).

[8] Patel, Y.S., Townend, P., Singh, A. et al. Modeling the Green Cloud Continuum: integrating energy considerations into Cloud–Edge models. *Cluster Computing (2024)*. https://doi.org/10.1007/s10586-024-04383-w

[9] [PREDICTION] Kumar, K. & Umamaheswari, E.. (2018). Prediction methods for effective resource provisioning in cloud computing: A survey. *Multiagent and Grid Systems*. 14. 283-305. 10.3233/MGS-180292.

[10] [ProactiveAutoScaler] A. Heimerson, J. Eker and K. -E. Årzén, "A Proactive Cloud Application Auto-Scaler using Reinforcement Learning," *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, Vancouver, WA, USA, 2022, pp. 213-220, doi: 10.1109/UCC56403.2022.00040.

[11] [HUNTER] Tuli, S., Gill, S. S., Xu, M., Garraghan, P., Bahsoon, R., Dustdar, S., Sakellariou, R., Rana, O., Buyya, R., Casale, G., & Jennings, N. R. (2022). HUNTER: AI based holistic resource management for sustainable cloud computing. *Journal of Systems and Software/ the Journal of Systems and Software*, 184, 111124. https://doi.org/10.1016/j.jss.2021.111124

[12] [SELFADAPTIVE] M. Xu, A. N. Toosi and R. Buyya, "A Self-Adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing," in *IEEE Transactions on Sustainable Computing*, vol. 6, no. 4, pp. 544-558, 1 Oct.-Dec. 2021, doi: 10.1109/TSUSC.2020.3014943.

[13] [HunterPlus] Iftikhar, S., Ahmad, M. M. M., Tuli, S., Chowdhury, D., Xu, M., Gill, S. S., & Uhlig, S. (2023). HunterPlus: AI based energy-efficient task scheduling for cloud–fog computing environments. *Internet of Things*, 21, 100667, Elsevier. https://doi.org/10.1016/j.iot.2022.100667

[14] [ANIARA] Sebastian Fredriksson, Lackis Eleftheriadis, Rickard Brännvall, Nils Bäckman, and Jonas Gustafsson. 2023. ANIARA: Experimental Investigation of Micro Edge Data Centers with Battery Support on Power-Constrained Grids. In *Companion Proceedings of the 14th ACM International Conference on Future Energy Systems (e-Energy '23 Companion)*. ACM, 72–78. https://doi.org/10.1145/3599733.3600252

[15] [CarbonAware] A. Radovanović et al., "Carbon-Aware Computing for Datacenters," in *IEEE Transactions on Power Systems*, vol. 38, no. 2, pp. 1270-1280, March 2023, doi: 10.1109/TPWRS.2022.3173250.

[16] [PowerModeling] A. Radovanovic, B. Chen, S. Talukdar, B. Roy, A. Duarte and M. Shahbazi, "Power Modeling for Effective Datacenter Planning and Compute Management," in *IEEE Transactions on Smart Grid*, vol. 13, no. 2, pp. 1611-1621, March 2022, doi: 10.1109/TSG.2021.3125275.

[17] [HolisticController] Albin Heimerson, Rickard Brännvall, Johannes Sjölund, Johan Eker, and Jonas Gustafsson. 2021. Towards a Holistic Controller: Reinforcement Learning for Data Center Control. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems (e-Energy '21)*. ACM. https://doi.org/10.1145/3447555.3466581